UNIVERSIDADE DE ÉVORA

5º Semestre
Ano Letivo 2020/2021

Inteligência Artificial

3º Trabalho Prático

Jogos de dois jogadores - Jogos com informação completa determinísticos

Trabalho realizado por:

Pedro Grilo, 43012

Diogo Castanho, 42496

Rodrigo Vicente, 42844

Introdução

Com a resolução deste trabalho é pretendido que consigamos representar dois jogos diferentes – jogo do Nim e jogo dos 4 em linha – com as estruturas de dados que achemos necessárias, sempre como um problema de pesquisa no espaço de estados.

Resolução dos Exercícios Propostos

1º Exercício

a) A estrutura de dados para os estados baseia-se na existência de um tuplo com 4 valores

```
%a)
%estado_inicial(e(0,0,1,2)). %representar as filas que temos
estado_inicial(e(1,3,5,7)). %exemplo da wikipedia
```

b) O estado terminal para este caso é o seguinte, também um tuplo com 4 valores

```
%b)
terminal(e(0,0,0,0)).
```

c) A função de utilidade para o estado terminal é a seguinte:

```
%c) Definir a função de utilidade (usando minimax que se chama a função valor)

valor(E,-1,P) :- terminal(E), R is P mod 2, R=1. %se a profundidade for impar, o computadorm perdeu e nós ganhamos

valor(E,1,P) :- terminal(E), R is P mod 2, R=0.  %se a profundidade for par, o computador ganhou e nós perdemos
```

Onde se a profundidade for ímpar, quer dizer que o computador perde e nós jogadores ganhámos.

Se a profundidade for par, o computador ganhou e nós perdemos.

d) Apesar de termos a implementação do minimax correta, por alguma razão não conseguimos escolher a melhor jogada com o mesmo.

e) Conseguimos implementar o algoritmo Alfa-Beta, mas para o problema do nim por alguma razão não deu nenhum resultado, mas sabendo só por si o que faz cada um dos algoritmos podemos afirmar que o algoritmo minimax demora muito mais tempo que o algoritmo do alfa-beta. Este acréscimo de tempo deve-se ao facto de este algoritmo visitar mais nós, com o corte alfa-beta resolve esse problema, visitando menos nós.

Instruções para compilar o problema do nim:

1. Abrir o terminal na pasta com os ficheiros
2. Abrir prolog
3. Compilar [minimax]
4. Compilar [nim]
5. Fazer joga.

2º Exercício

a) A estrutura de dados escolhida foi uma lista de listas, em que cada lista representa uma linha do tabuleiro.

```
% cada posicao pode ter "x", "o" ou "v" (vazio)
estado_inicial([[v,v,v,v,v],
                [v,v,o,x,v],
                [x,o,x,x,v],
                [o,x,x,x,o]]).
```

b) O predicado terminal é o seguinte, onde um estado é terminal se houver uma sequência de 4 “x” ou 4 “o” seguidos, quer em numa linha horizontal, vertical ou diagonal.

```
terminal(G) :- linhas(G,_).
terminal(G) :- colunas(G,_).
terminal(G) :- diagonal(G,_).
terminal(G) :- cheio(G).

linhas([[X,X,X,X,_],_,_,_],X) :- X \= v.
linhas([[_,X,X,X,X],_,_,_],X) :- X \= v.

linhas([_,[X,X,X,X,_],_,_],X) :- X \= v.
linhas([_,[_,X,X,X,X],_,_],X) :- X \= v.

linhas([_,_,[X,X,X,X,_],_],X) :- X \= v.
linhas([_,_,[_,X,X,X,X],_],X) :- X \= v.

linhas([_,_,_,[X,X,X,X,_]],X) :- X \= v.
linhas([_,_,_,[_,X,X,X,X]],X) :- X \= v.

colunas([[X,_,_,_,_],[X,_,_,_,_],[X,_,_,_,_], [X,_,_,_,_]],X) :- X \= v.

colunas([[_,X,_,_,_],[_,X,_,_,_],[_,X,_,_,_], [_,X,_,_,_]],X) :- X \= v.

colunas([[_,_,X,_,_],[_,_,X,_,_],[_,_,X,_,_], [_,_,X,_,_]],X) :- X \= v.

colunas([[_,_,_,X,_],[_,_,_,X,_],[_,_,_,X,_], [_,_,_,X,_]],X) :- X \= v.

colunas([[_,_,_,_,X],[_,_,_,_,X],[_,_,_,_,X], [_,_,_,_,X]],X) :- X \= v.
```

```
diagonal([[X,_,_,_,_],[_,X,_,_,_],[_,_,X,_,_], [_,_,_,X,_]],X) :- X \= v.
diagonal([[_,_,_,X,_],[_,_,X,_,_],[_,X,_,_,_], [X,_,_,_,_]],X) :- X \= v.

diagonal([[_,X,_,_,_],[_,_,X,_,_],[_,_,_,X,_], [_,_,_,_,X]],X) :- X \= v.
diagonal([[_,_,_,_,X],[_,_,_,X,_],[_,_,X,_,_], [_,X,_,_,_]],X) :- X \= v.


cheio([L1,L2,L3,L4]) :-
    append(L1,L2, L12),
    append(L12, L3, L123),
    append(L123, L4, L1234),
    \+ member(v, L1234).
```

c) A função de utilidade definida recebe um estado terminal e devolve -1 em caso de derrota para o computador, 1 para vitória do jogador (nós) e 0 caso seja empate.

```
%função de utilidade, retorna o valor dos estados terminais, 1 ganha -1 perde
valor(G, 1) :- linhas(G,x).
valor(G, 1) :- colunas(G,x).
valor(G, 1) :- diagonal(G,x).
valor(G, -1) :- linhas(G,o).
valor(G, -1) :- colunas(G,o).
valor(G, -1) :- diagonal(G,o).
valor(_, 0).
```

d) O agente inteligente está implementado nas implementações minimax.pl e alfabeta.pl.
Para se usar, basta no decorrer do jogo fazer a nossa jogada, inserindo a coluna onde queremos colocar a peça, e depois o agente inteligente irá fazer a sua jogada consoante a nossa.

e) O algoritmo Minimax vai visitar um número muito superior de estados, e por isso o seu tempo de compilação vai ser superior.

Como exemplo, com o estado inicial dado, e colocando nós jogador a peça na coluna 4, o jogo terminará logo, demorando com alfabeta cerca de 1ms a chegar ao resultado.
Mas, já com minimax, o mesmo resultado (rápido e eficaz na mesma, leva cerca de 5ms).

Devido a erros de full stack overlflow, alguns exemplos não irão funcionar, tais como, quando o tabuleiro está vazio no início, entre outros com combinações diversas existentes.
O estado inicial dado tem resultado onde quer que nós coloquemos a nossa peça em primeiro lugar, pois ganhado logo, ou perdendo, o jogo terminará (perdendo uma vez que o computador é inteligente e irá colocar a sua peça para ganhar na coluna em que o pode fazer).

<u>Instruções para compilar o problema do sudoku:</u>

6. Abrir o terminal na pasta com os ficheiros
7. Abrir prolog
8. Compilar [alfabeta] para algoritmo com corte alfabeta, e [minimax] para algoritmo minimax.
9. Depois temos de fazer [emlinha4]. Para compilar o jogo
10. Fazer joga. para ser mostrado o tabuleiro
11. Escolher a coluna onde colocar a peça, exemplo: 2.
12. Depois o computador faz a sua joga, sendo que pode demorar alguns bons milissegundos a executar a sua jogada.

Conclusão

Com a realização deste trabalho ficámos a ter mais conhecimento sobre a resolução de problemas como um problema de pesquisa no espaço de estados, pois aprofundámos e usámos uma vertente mais prática dos problemas dados nas aulas.

Para além disso, foi interessante a escolha de um jogo de dois jogadores feito por nós, pois tinha de ser um adequado ao nosso saber da matéria, sendo o 4 em linha, na nossa opinião, adequado ao mesmo.

Pensamos ter atingido a maioria dos objetivos do trabalho, principalmente para o problema do jogo do 4 em linha, onde apesar de termos stack overflow em alguns tabuleiros, pensamos ter atingido (e percebido) a totalidade das perguntas do mesmo.

Para o primeiro problema do nim, tivemos bastante mais dificuldades, não conseguindo fazer algumas perguntas pedidas no enunciado.